Human Resource Management
Predicting Employee Promotions using Machine Learning

Team ID: SWTID1749627644

Team member 1: Avirup Kundu

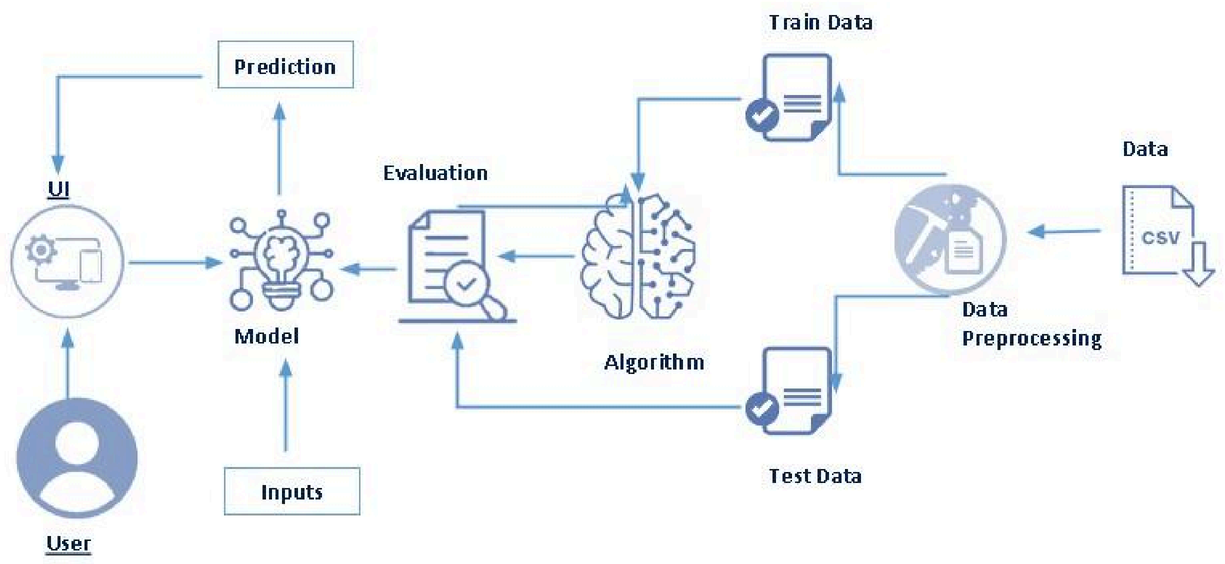
Team member 2: Lokeshwer Vijayakumar

Team member 3: Vishwesh Menon

Employee Promotion Prediction using Machine Learning

In modern organizations, especially large-scale enterprises and fast-growing startups, identifying the right employees for promotion is often challenging due to the volume and complexity of workforce data. Traditional methods can be subjective, inconsistent, and time-consuming, leading to missed opportunities for recognizing high-potential talent. This project aims to develop a machine learning model that can accurately predict the likelihood of employee promotions based on factors such as performance ratings, training scores, tenure, awards, and more. By automating and optimizing the promotion decision process, the model supports fair, data-driven human resource management that enhances employee engagement, improves retention, and drives organizational growth.

Technical Architecture



Project Flow:

- The user interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
    - Collect the dataset or create the dataset
- Visualizing and analyzing data
    - Univariate analysis
    - Multivariate analysis
    - Descriptive analysis
- Data pre-processing
    - Drop unwanted features
    - Checking for null values
    - Remove negative data
    - Handling outlier
    - Handling categorical data
    - Handling Imbalanced data
    - Splitting data into train and test
- Model building
    - Import the model building libraries
    - Initializing the model
    - Training and testing the model
    - Evaluating performance of the model
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code
    - Run the Application

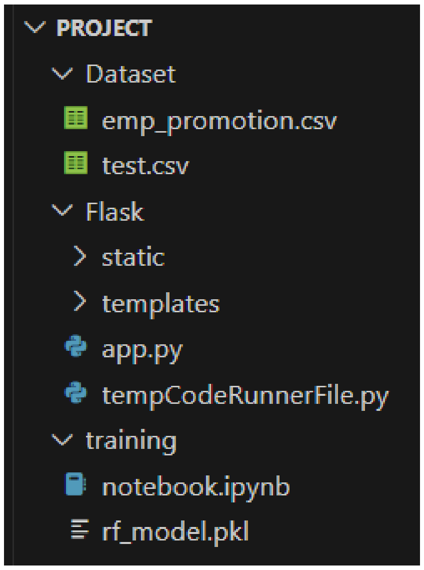
Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- ML Concepts:
  - Supervised and Unsupervised learning: https://www.javatpoint.com/supervised-machine-learning https://www.javatpoint.com/unsupervised-machine-learning
  - Regression Classification and Clustering: https://youtu.be/6za9_mh3uTE
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classificationalgorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-tounderstand-the-math-behind-xgboost/
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-modelevaluation-error-metrics/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



```
PROJECT
  Dataset
    emp_promotion.csv
    test.csv
  Flask
    static
    templates
    app.py
    tempCodeRunnerFile.py
  training
    notebook.ipynb
    rf_model.pkl
```

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- rf_model.pkl is our saved model. Further we will use this model for flask integration
- Dataset Folder contains the Dataset used
- The notebook file contains procedures for building the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

The core business problem is the challenge organizations face in accurately and fairly identifying employees who are deserving of promotions. Traditional methods often involve manual reviews, which can be time-consuming, biased, or overlook deserving candidates. This project seeks to automate and optimize the promotion prediction process using machine learning by analyzing key performance indicators and employee data.

Activity 2: Business Requirements

Key business requirements for the employee promotion prediction system include:

- ❖ Accuracy & Fairness: The model must ensure fair evaluation based on objective performance and historical promotion trends.
- ❖ Scalability: It should be scalable to accommodate data from large organizations with thousands of employees.
- ❖ Interpretability: The system should allow HR teams to understand the reasoning behind predictions.
- ❖ Integration Friendly: The model should be deployable into existing HR tools or dashboards.
- ❖ Data Security & Privacy: Must comply with company and data protection regulations, safeguarding employee data.

Activity 3: Literature Survey

Machine learning is increasingly being applied in Human Resource Management to facilitate data-based decisions for hiring, performance appraisal, and promotion. Decision Trees, Random Forests, and Gradient Boosting models have been found in certain studies to effectively predict promotions based on variables such as tenure, training score, performance rating, and appraisals in the past.

Statistics show ensemble models to be more robust and accurate than simple models. Sharma et al. (2021), for example, found Random Forests to be highly effective in identifying high-potential employees. Interpretability and fairness have now become a concern, with controls to render the models used in HR transparent and non-discriminatory instituted.

Kaggle and UCI open data have made mass experimentation possible, and we have proven that machine learning can provide accurate and scalable solutions for employee promotion prediction. However, bias, explainability, and ethics must be properly addressed when applying such models to actual HR systems.

Activity 4: Social or Business Impact

Social Impact: Improved fairness and transparency in promotion decisions can lead to higher employee satisfaction, better retention rates, and trust in the organization’s HR processes.

Business Impact: Helps streamline HR processes, reduce manual workload, and make data-driven decisions to promote deserving employees. This fosters a culture of meritocracy, improves productivity, and aligns talent development with organizational goals.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

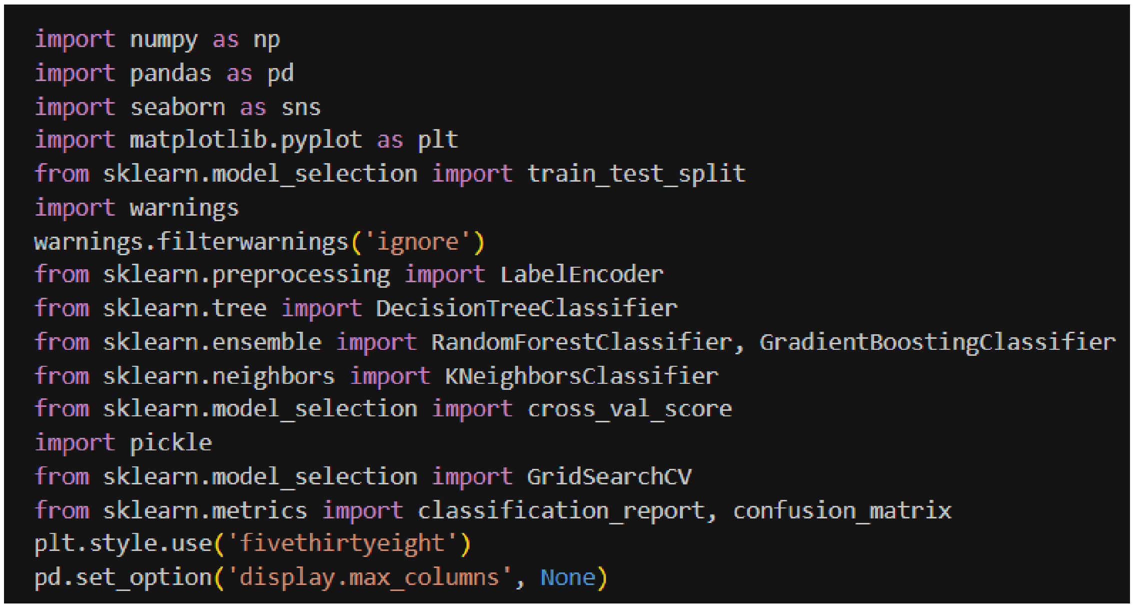
In this project, we have used emp_promotion.csv data. This data is downloaded from Skill Wallet website. Please refer to the link given below to download the dataset.
https://drive.google.com/file/d/14eQR1VWHwuomPaXdKuIkZEpSstvav8Db/view?usp=sharing

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

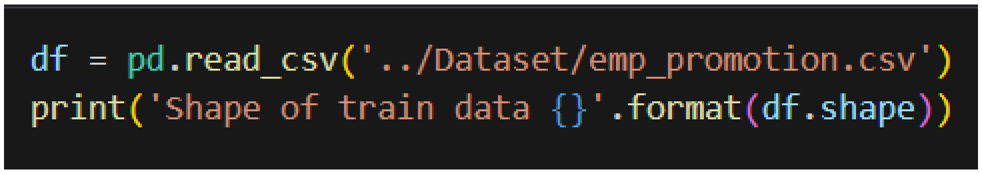
Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.



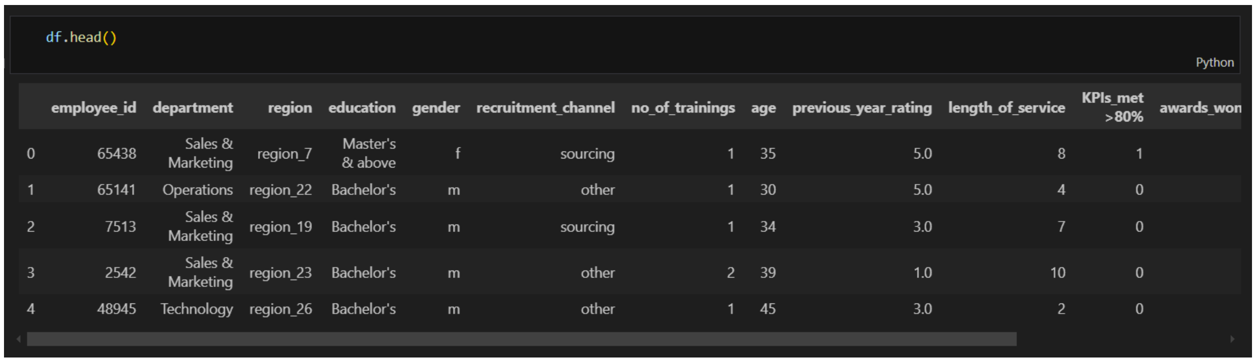
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import pickle
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
plt.style.use('fivethirtyeight')
pd.set_option('display.max_columns', None)
```

Activity 1.2: Read the Dataset

- Our dataset format might be in .csv, excel files, .txt, .json, etc.
- We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.



```python
df = pd.read_csv('../Dataset/emp_promotion.csv')
print('Shape of train data {}'.format(df.shape))
```



```python
df.head()
```

Python

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1	35	5.0	8	1	
1	65141	Operations	region_22	Bachelor's	m	other	1	30	5.0	4	0	
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1	34	3.0	7	0	
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2	39	1.0	10	0	
4	48945	Technology	region_26	Bachelor's	m	other	1	45	3.0	2	0	

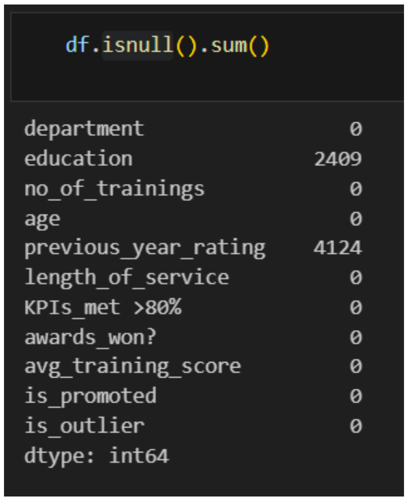
Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data. The downloaded data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps:
- Handling Missing Data
- Handling Outliers
- Handling Irrelevant Columns

Activity 2.1: Handling Missing Data

For checking missing data, df.isnull() function is used. To sum those values feature wise we use .sum() function

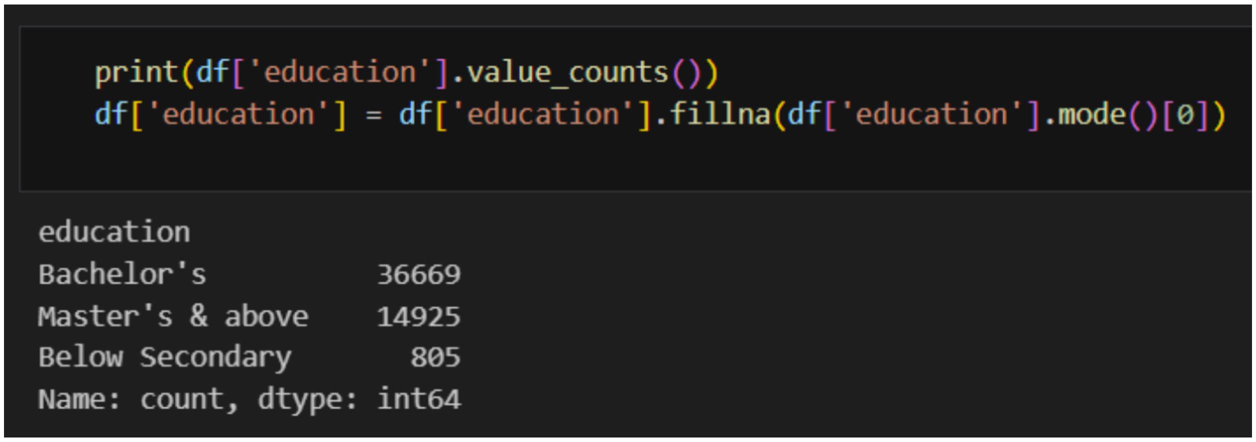


```
df.isnull().sum()
```

```
department                 0
education               2409
no_of_trainings            0
age                        0
previous_year_rating    4124
length_of_service          0
KPIs_met >80%              0
awards_won?                0
avg_training_score         0
is_promoted                0
is_outlier                 0
dtype: int64
```

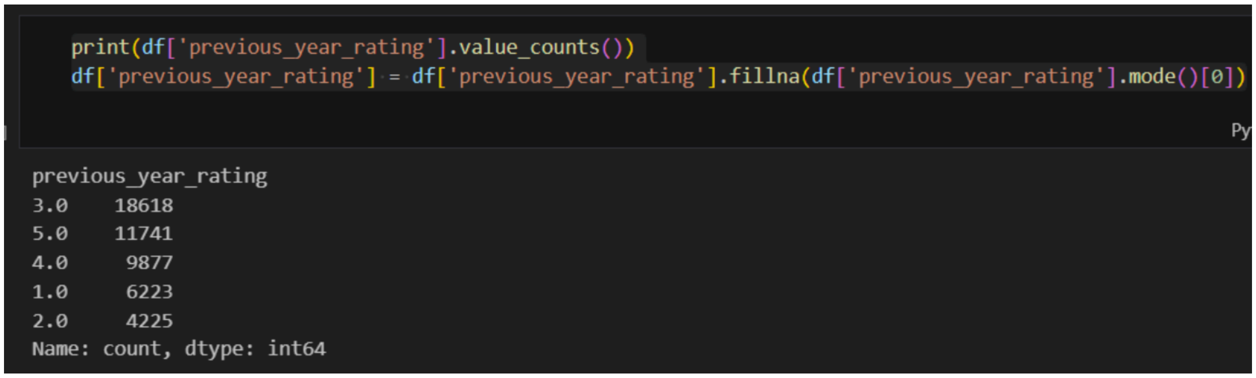
We can see that there are a lot of null values in education and previous_year_rating columns. Let's handle those.

We will first find how many times each unique value appears in the education column. Then we will get the most frequent value (mode) from the data and use fillna() function to fill the missing values with the mode.



```
print(df['education'].value_counts())
df['education'] = df['education'].fillna(df['education'].mode()[0])
```

```
education
Bachelor's          36669
Master's & above    14925
Below Secondary       805
Name: count, dtype: int64
```

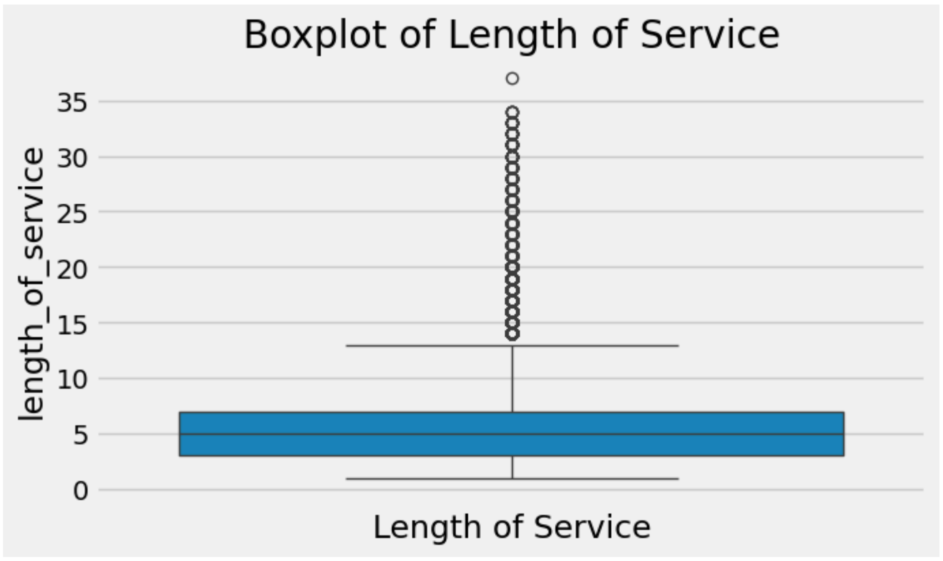


```
print(df['previous_year_rating'].value_counts())
df['previous_year_rating'] = df['previous_year_rating'].fillna(df['previous_year_rating'].mode()[0])
```

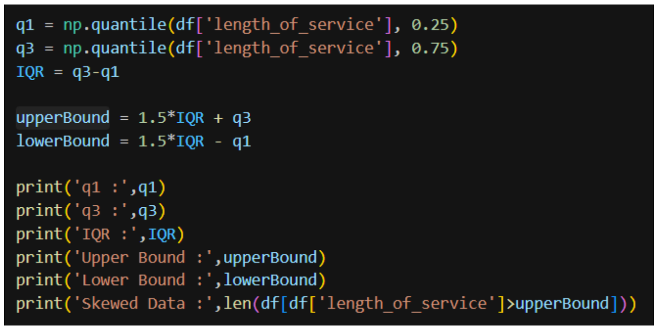
```
previous_year_rating
3.0    18618
5.0    11741
4.0     9877
1.0     6223
2.0     4225
Name: count, dtype: int64
```

Activity 2.2: Handling Outliers

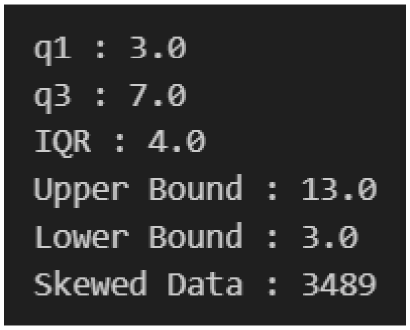
We start by visualizing the distribution of the length_of_service feature using a boxplot to easily spot outliers.



We calculate the Interquartile Range (IQR) and define the upper bound for outliers. To find the upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find the lower bound instead of adding, subtract it with 1st quantile. Take the image attached below as your reference.

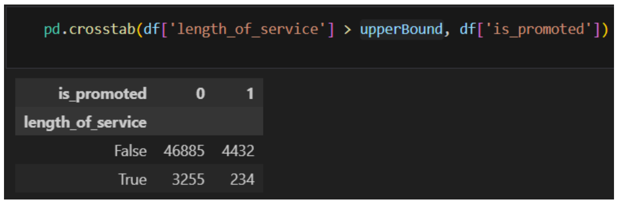


```
q1 = np.quantile(df['length_of_service'], 0.25)
q3 = np.quantile(df['length_of_service'], 0.75)
IQR = q3-q1

upperBound = 1.5*IQR + q3
lowerBound = 1.5*IQR - q1

print('q1 :',q1)
print('q3 :',q3)
print('IQR :',IQR)
print('Upper Bound :',upperBound)
print('Lower Bound :',lowerBound)
print('Skewed Data :',len(df[df['length_of_service']>upperBound]))
```



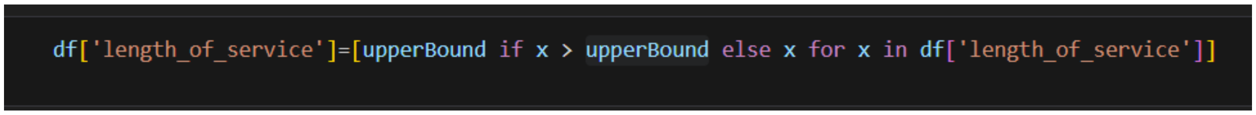
```
q1 : 3.0
q3 : 7.0
IQR : 4.0
Upper Bound : 13.0
Lower Bound : 3.0
Skewed Data : 3489
```

Since removing these data points may not be ideal, we cap the outlier values at the upper bound to reduce their influence.



```
pd.crosstab(df['length_of_service'] > upperBound, df['is_promoted'])
```

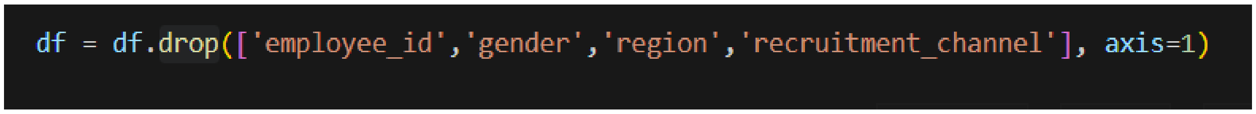
is_promoted	0	1
length_of_service		
False	46885	4432
True	3255	234



```
df['length_of_service']=[upperBound if x > upperBound else x for x in df['length_of_service']]
```

Activity 2.3: Handling Irrelevant Columns

We do not need employee id, gender, region, recruitment channel attributes for predicting promotion, so we will be removing these unwanted features.

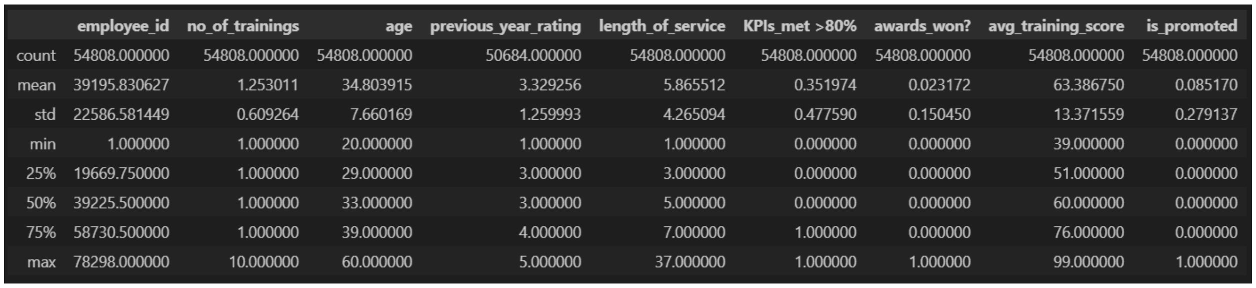


```
df = df.drop(['employee_id','gender','region','recruitment_channel'], axis=1)
```

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistics

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features



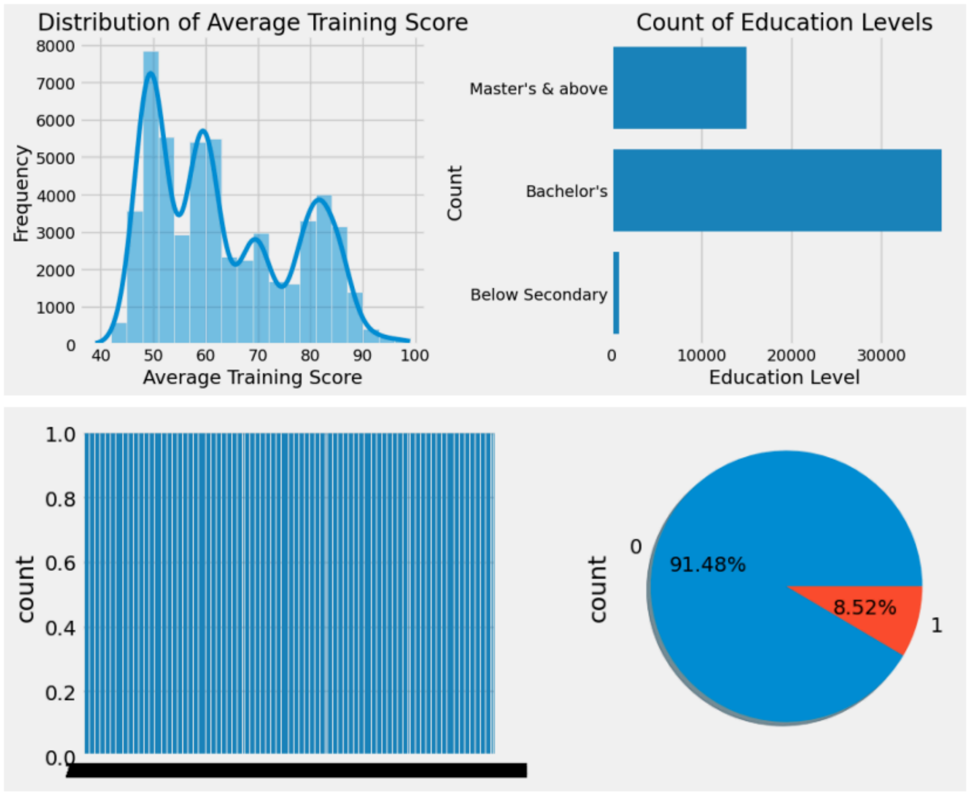
	employee_id	no_of_trainings	age	previous_year_rating	length_of_service	KPIs_met >80%	awards_won?	avg_training_score	is_promoted
count	54808.000000	54808.000000	54808.000000	50684.000000	54808.000000	54808.000000	54808.000000	54808.000000	54808.000000
mean	39195.830627	1.253011	34.803915	3.329256	5.865512	0.351974	0.023172	63.386750	0.085170
std	22586.581449	0.609264	7.660169	1.259993	4.265094	0.477590	0.150450	13.371559	0.279137
min	1.000000	1.000000	20.000000	1.000000	1.000000	0.000000	0.000000	39.000000	0.000000
25%	19669.750000	1.000000	29.000000	3.000000	3.000000	0.000000	0.000000	51.000000	0.000000
50%	39225.500000	1.000000	33.000000	3.000000	5.000000	0.000000	0.000000	60.000000	0.000000
75%	58730.500000	1.000000	39.000000	4.000000	7.000000	1.000000	0.000000	76.000000	0.000000
max	78298.000000	10.000000	60.000000	5.000000	37.000000	1.000000	1.000000	99.000000	1.000000

Activity 2: Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate Analysis

Univariate analysis is understanding the data with a single feature.



Distribution of Average Training Score:

- The histogram shows the frequency distribution of employees' average training scores.
- The distribution is slightly right-skewed, with most employees scoring between 45 and 85.
- There are multiple peaks, suggesting the presence of subgroups or clusters in training performance.
- Very few employees have extremely low or extremely high training scores.

Count of Education Levels:

- The bar plot displays the count of employees by education level.
- The majority of employees have a Bachelor's degree.
- A significant portion holds a Master's degree or higher.
- Very few employees have education below secondary level.

- This indicates that the workforce is generally well-educated, with higher education levels being more common.

Promotion Class Distribution:

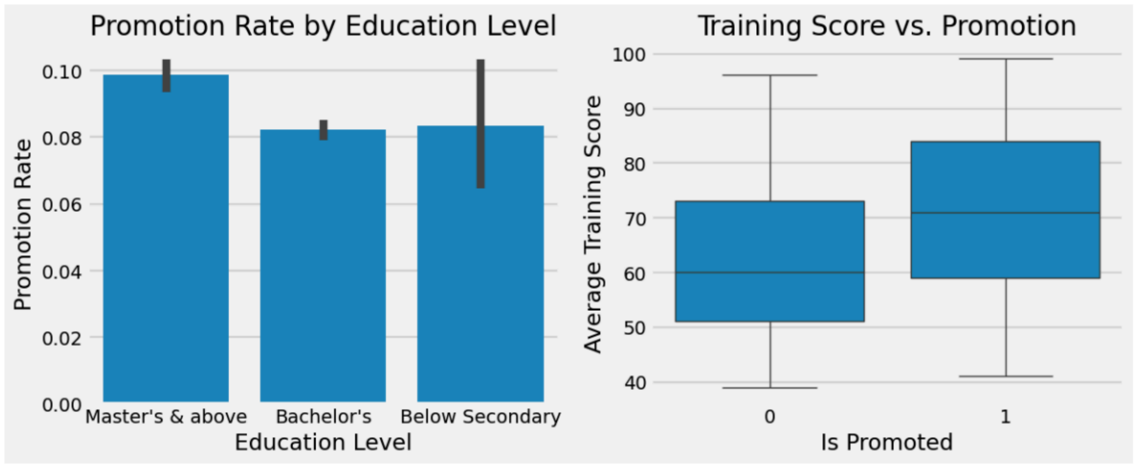
- The count plot shows a significant class imbalance in the dataset.
- The vast majority of employees were not promoted (class 0), while only a small fraction were promoted (class 1).
- This imbalance is visually evident, with the bar for class 0 being much taller than for class 1.

Promotion Class Distribution:

- The pie chart quantifies the imbalance: 91.48% of employees were not promoted, while only 8.52% received a promotion.
- This highlights the need for special handling (such as resampling) during model training to avoid bias toward the majority class.
- The data suggests that promotions are relatively rare events in this organization.

Activity 2.2: Bivariate Analysis

To find the relation between two features we use bivariate analysis



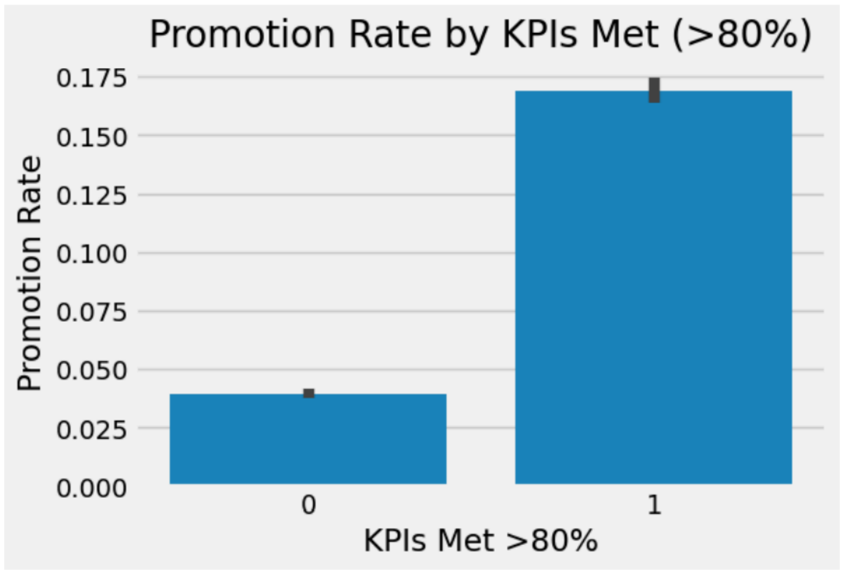
Promotion Rate by Education Level:

- Employees with a Master's degree or above have the highest promotion rate among all education levels.
- Those with a Bachelor's degree and Below Secondary education have slightly lower and similar promotion rates.

- This suggests that higher education may provide a slight advantage for promotion, but the difference is not very large.

Training Score vs. Promotion:

- The boxplot shows that employees who were promoted generally have higher average training scores compared to those who were not promoted.
- The median training score for promoted employees is noticeably higher.
- Promoted employees also tend to have a wider range of training scores, but overall, higher training performance is associated with a greater chance of promotion.

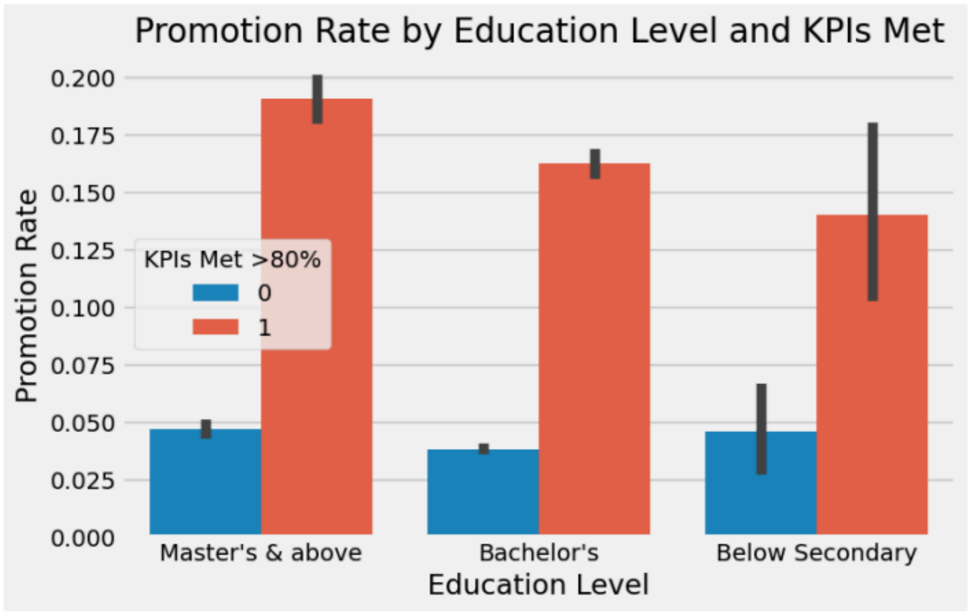


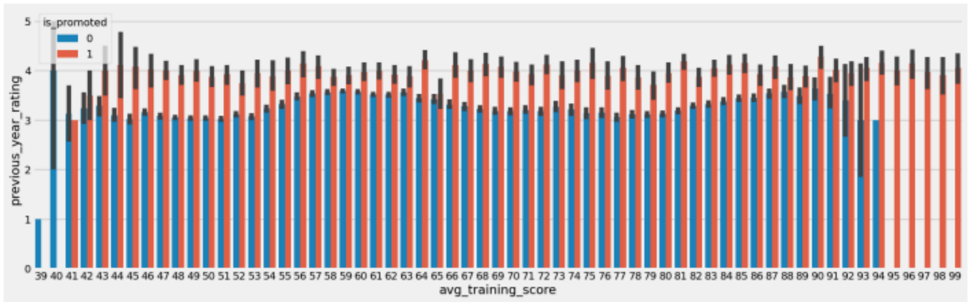
Promotion Rate by KPIs Met (>80%):

- Employees who met more than 80% of their KPIs have a significantly higher promotion rate compared to those who did not.
- The promotion rate for employees meeting KPIs >80% is more than four times higher than for those who did not meet this threshold.
- This indicates that meeting key performance indicators is a strong factor influencing promotion decisions in the organization.

Activity 2.3: Multivariate Analysis

Multivariate analysis is to find the relation between multiple features.





Promotion Rate by Education Level and KPIs Met:

- For all education levels, employees who met more than 80% of their KPIs have a much higher promotion rate than those who did not.
- Among those who met KPIs >80%, employees with a Master's & above have the highest promotion rate, followed by Bachelor's and Below Secondary.
- For employees who did not meet KPIs >80%, the promotion rate is low across all education levels.
- This shows that meeting KPIs is a key driver for promotion, and higher education provides an additional, but smaller, advantage.

Previous Year Rating vs. Average Training Score by Promotion Status:

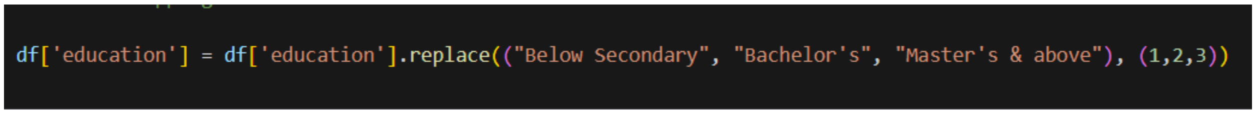
- Employees who were promoted (orange bars) generally have higher previous year ratings across almost all average training score levels.
- For higher training scores (above ~90), nearly all employees with high previous year ratings were promoted, indicating a strong link between high training performance, good past ratings, and promotion.

- Employees with lower training scores and lower previous year ratings are much less likely to be promoted.
- The plot highlights that both a high average training score and a high previous year rating are important factors for promotion.

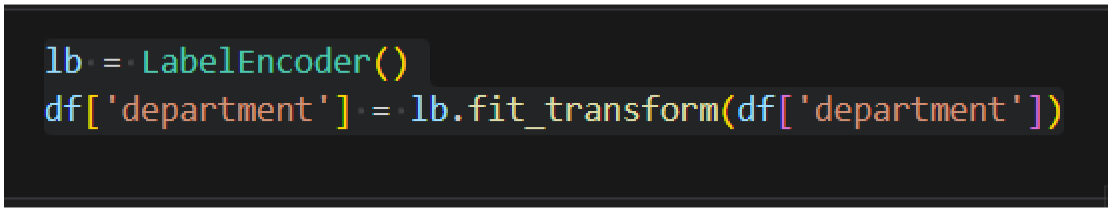
Encoding Categorical Values

Converting categorical (text) data into numerical values so that machine learning models can process them.

- education is mapped to 1, 2, 3 for its categories.
- department is label-encoded to assign a unique integer to each department.



```
df['education'] = df['education'].replace(("Below Secondary", "Bachelor's", "Master's & above"), (1,2,3))
```

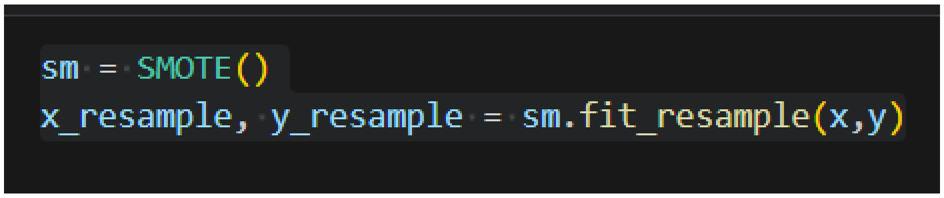


```
lb = LabelEncoder()
df['department'] = lb.fit_transform(df['department'])
```

Handling Imbalanced Data

Balancing the number of samples in each class (promoted/not promoted) to prevent bias in model training.

- SMOTE (Synthetic Minority Over-sampling Technique) is used to generate synthetic samples for the minority class.

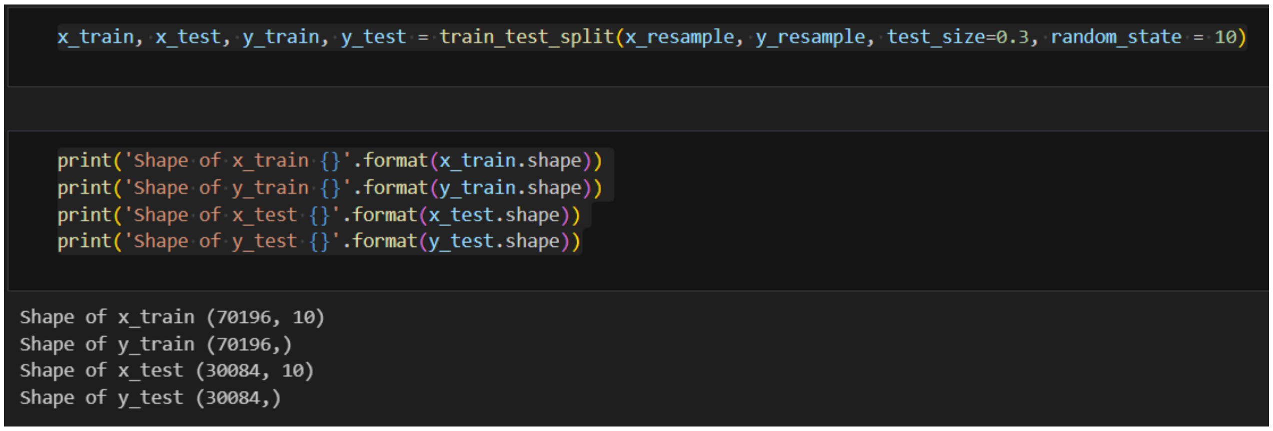


```
sm = SMOTE()
x_resample, y_resample = sm.fit_resample(x,y)
```

Splitting data into train and test

Dividing the dataset into training and testing sets to evaluate model performance on unseen data.

- 70% of the data is used for training, 30% for testing, after balancing with SMOTE.



```
x_train, x_test, y_train, y_test = train_test_split(x_resample, y_resample, test_size=0.3, random_state = 10)
```

```
print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_test.shape))
print('Shape of y_test {}'.format(y_test.shape))
```

```
Shape of x_train (70196, 10)
Shape of y_train (70196,)
Shape of x_test (30084, 10)
Shape of y_test (30084,)
```

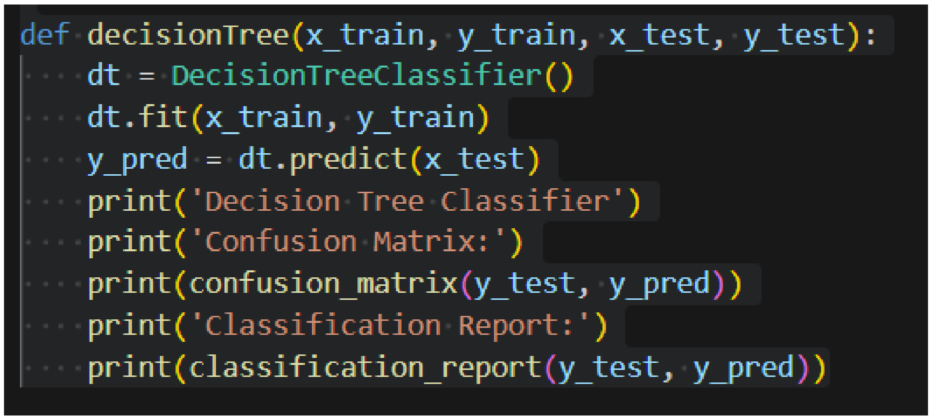
Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on the performance.

Activity 1.1: Decision Tree Model

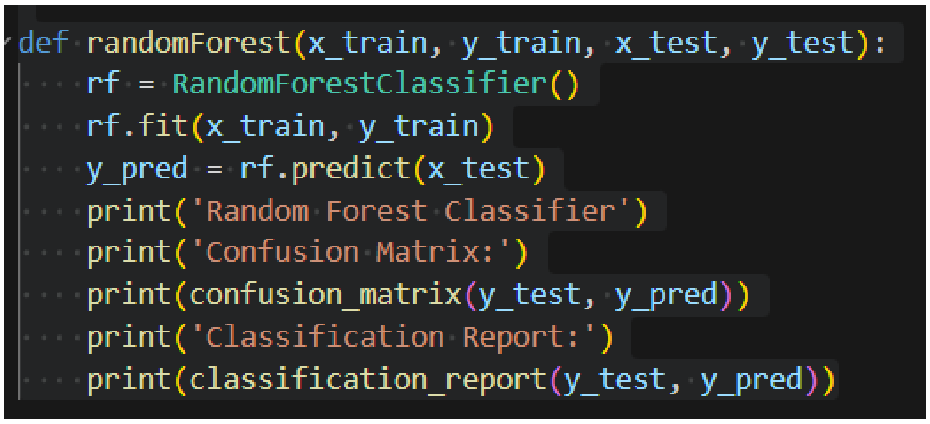
The Decision Tree Classifier is imported from 'sklearn.tree'. An instance of 'DecisionTreeClassifier' is created and trained using the '.fit()' method on the training data. The model then makes predictions on the test data using '.predict()'. Performance is evaluated using the confusion matrix and classification report. This model is easy to interpret and works well for structured data with clear decision boundaries.



```
def decisionTree(x_train, y_train, x_test, y_test):
    dt = DecisionTreeClassifier()
    dt.fit(x_train, y_train)
    y_pred = dt.predict(x_test)
    print('Decision Tree Classifier')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.2: Random Forest Model

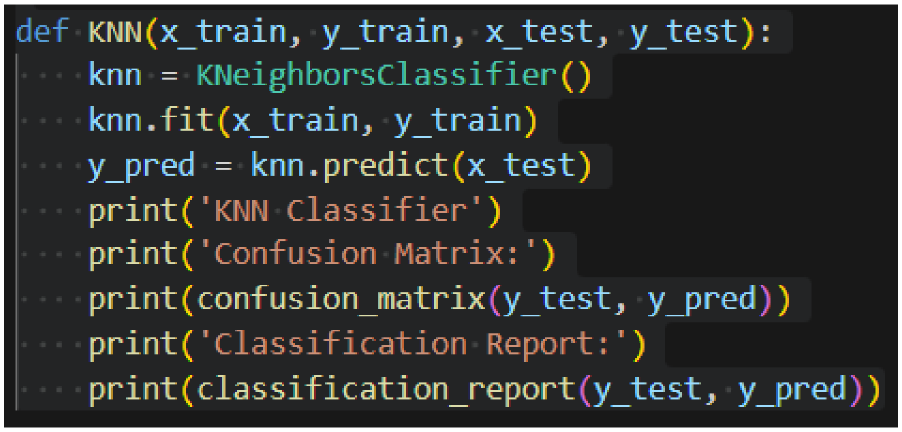
The Random Forest Classifier is imported from sklearn.ensemble. The model is initialized with default or tuned parameters and trained using .fit() on the training dataset. Predictions are made on the test set using .predict(). Random Forest works by combining multiple decision trees, leading to improved accuracy and robustness while reducing overfitting.



```python
def randomForest(x_train, y_train, x_test, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)
    y_pred = rf.predict(x_test)
    print('Random Forest Classifier')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.3: KNN Model

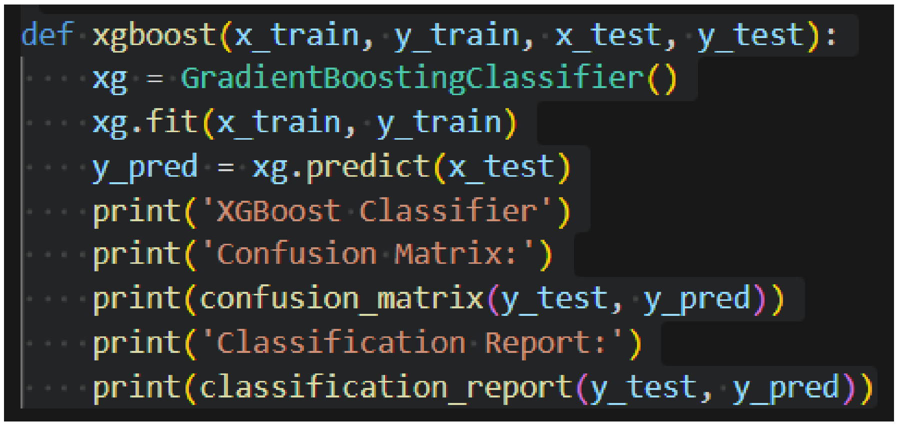
The KNN Classifier is imported from sklearn.neighbors. After creating the KNeighborsClassifier instance, the model is trained using .fit() and tested using .predict() on the test dataset. KNN predicts outcomes based on the majority label of the 'k' closest data points, making it simple and effective for balanced datasets.



```python
def KNN(x_train, y_train, x_test, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    print('KNN Classifier')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.4: XGBoost Model

The XGBoost model is implemented using GradientBoostingClassifier from sklearn.ensemble. The classifier is initialized and trained with .fit(), followed by prediction using .predict() on test data. This boosting algorithm improves model accuracy by sequentially minimizing errors from previous models, making it highly effective for structured/tabular datasets.



```python
def xgboost(x_train, y_train, x_test, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train, y_train)
    y_pred = xg.predict(x_test)
    print('XGBoost Classifier')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 2: Testing the model

We Can use the ‘predict()’ function to test each model.

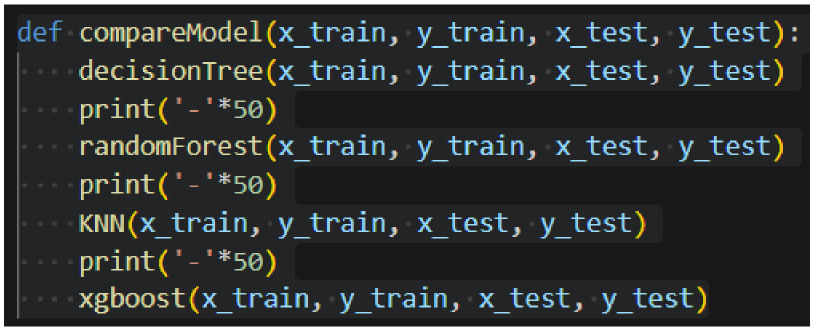
Milestone 5: Performance Testing and Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

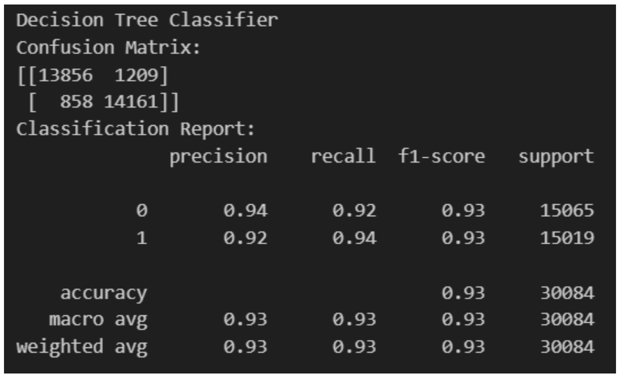
Here we have tested all the models and have their metrics (Confusion matrix and Classification Report). Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score

Compare the models

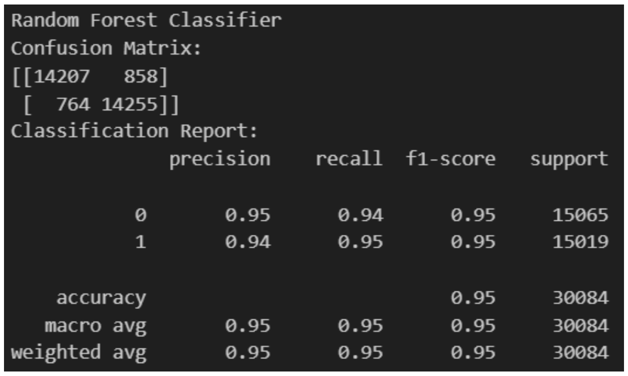
We have defined a comparison function for this purpose:



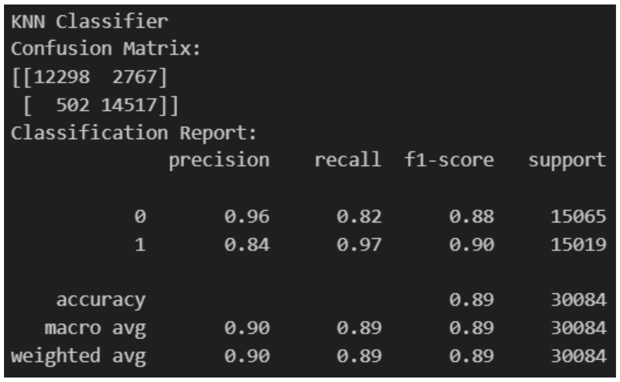
```
def compareModel(x_train, y_train, x_test, y_test):
    decisionTree(x_train, y_train, x_test, y_test)
    print('-'*50)
    randomForest(x_train, y_train, x_test, y_test)
    print('-'*50)
    KNN(x_train, y_train, x_test, y_test)
    print('-'*50)
    xgboost(x_train, y_train, x_test, y_test)
```



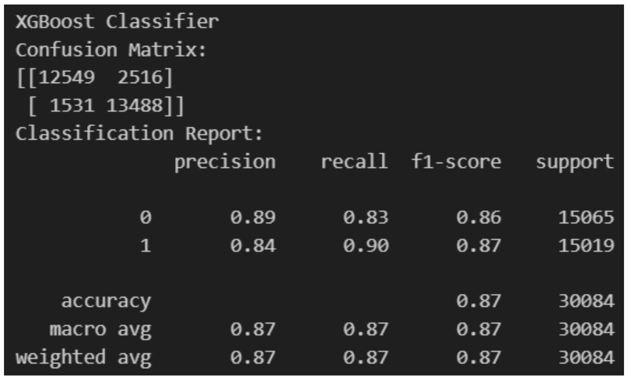
```
Decision Tree Classifier
Confusion Matrix:
[[13856  1209]
 [  858 14161]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     15065
           1       0.92      0.94      0.93     15019

    accuracy                           0.93     30084
   macro avg       0.93      0.93      0.93     30084
weighted avg       0.93      0.93      0.93     30084
```



```
Random Forest Classifier
Confusion Matrix:
[[14207   858]
 [  764 14255]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.95     15065
           1       0.94      0.95      0.95     15019

    accuracy                           0.95     30084
   macro avg       0.95      0.95      0.95     30084
weighted avg       0.95      0.95      0.95     30084
```



```
KNN Classifier
Confusion Matrix:
[[12298  2767]
 [  502 14517]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.82      0.88     15065
           1       0.84      0.97      0.90     15019

    accuracy                           0.89     30084
   macro avg       0.90      0.89      0.89     30084
weighted avg       0.90      0.89      0.89     30084
```



```
XGBoost Classifier
Confusion Matrix:
[[12549  2516]
 [ 1531 13488]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.83      0.86     15065
           1       0.84      0.90      0.87     15019

    accuracy                           0.87     30084
   macro avg       0.87      0.87      0.87     30084
weighted avg       0.87      0.87      0.87     30084
```

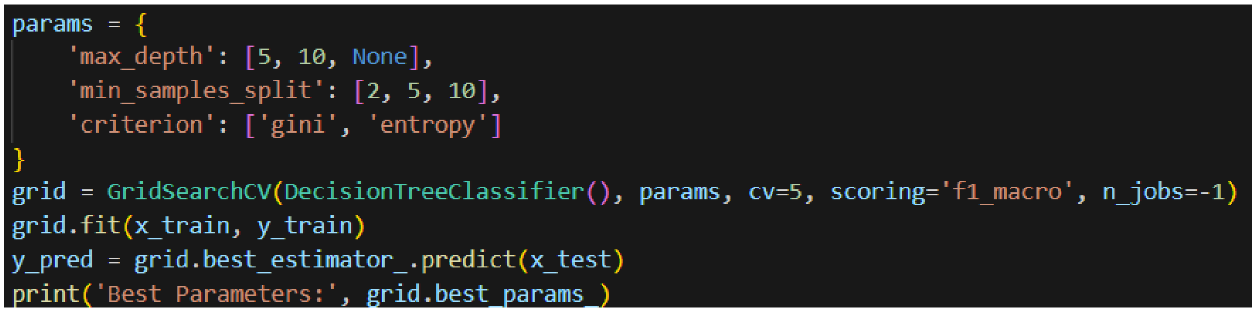
From the above models and evaluation, we see that the Random Forest model is the best performing model.

Activity 2: Apply hyperparameter tuning and compare model accuracy

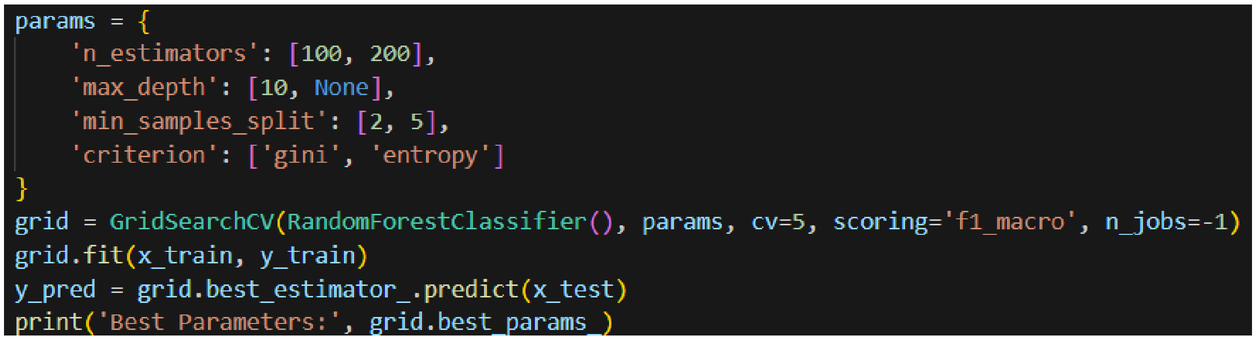
Hyperparameter tuning is performed using GridSearchCV from scikit-learn for each model.

- For each model, a parameter grid is defined (e.g., max_depth, n_estimators, min_samples_split, etc.).
- GridSearchCV systematically tries all combinations of these parameters using 5-fold cross-validation and selects the best combination based on the f1_macro score.

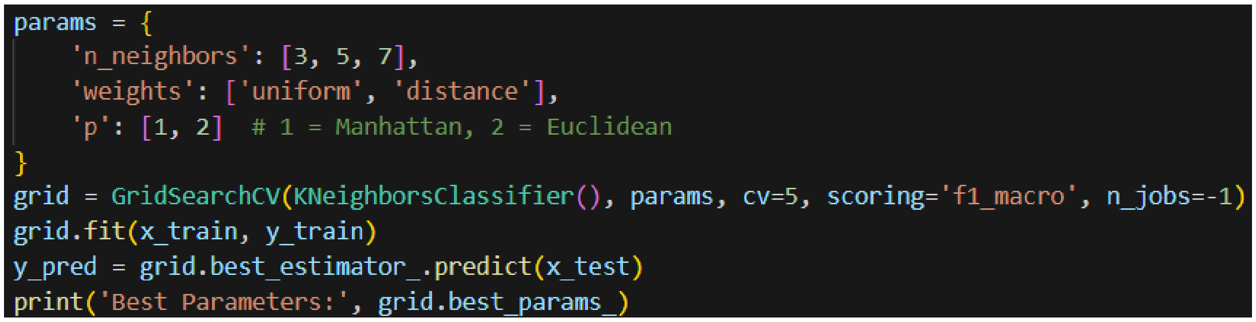
- The best estimator is then used to predict on the test set, and its performance is reported.



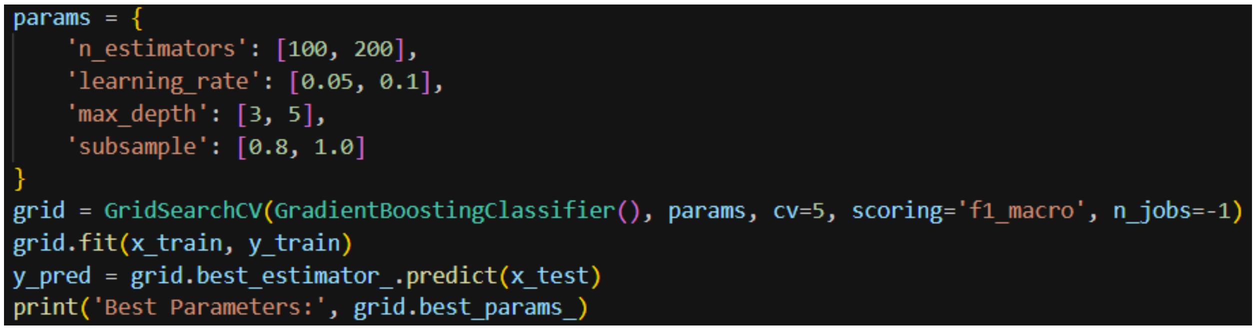
```
params = {
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}
grid = GridSearchCV(DecisionTreeClassifier(), params, cv=5, scoring='f1_macro', n_jobs=-1)
grid.fit(x_train, y_train)
y_pred = grid.best_estimator_.predict(x_test)
print('Best Parameters:', grid.best_params_)
```



```
params = {
    'n_estimators': [100, 200],
    'max_depth': [10, None],
    'min_samples_split': [2, 5],
    'criterion': ['gini', 'entropy']
}
grid = GridSearchCV(RandomForestClassifier(), params, cv=5, scoring='f1_macro', n_jobs=-1)
grid.fit(x_train, y_train)
y_pred = grid.best_estimator_.predict(x_test)
print('Best Parameters:', grid.best_params_)
```

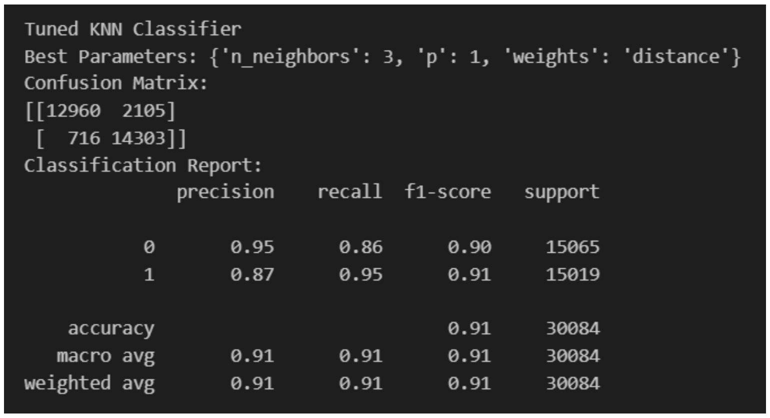


```
params = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]  # 1 = Manhattan, 2 = Euclidean
}
grid = GridSearchCV(KNeighborsClassifier(), params, cv=5, scoring='f1_macro', n_jobs=-1)
grid.fit(x_train, y_train)
y_pred = grid.best_estimator_.predict(x_test)
print('Best Parameters:', grid.best_params_)
```

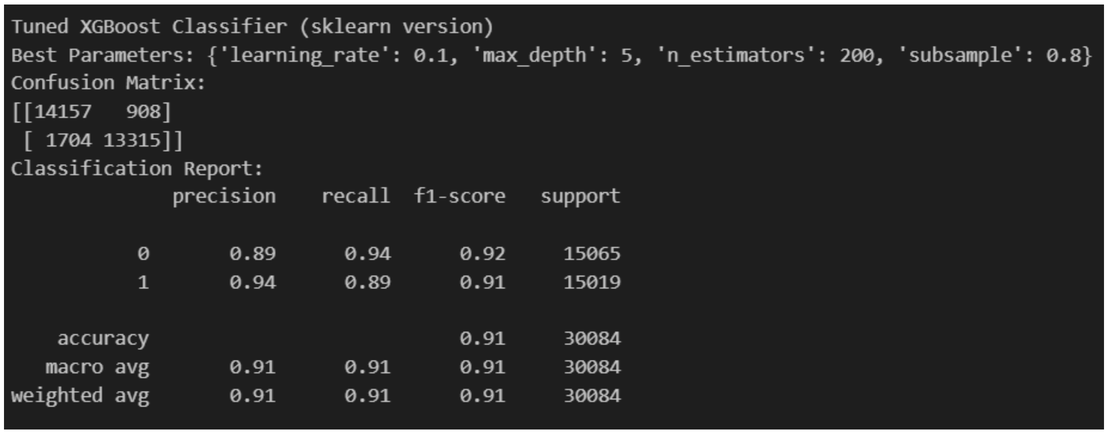


```
params = {
    'n_estimators': [100, 200],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]
}
grid = GridSearchCV(GradientBoostingClassifier(), params, cv=5, scoring='f1_macro', n_jobs=-1)
grid.fit(x_train, y_train)
y_pred = grid.best_estimator_.predict(x_test)
print('Best Parameters:', grid.best_params_)
```

The KNN Model and the XGBoost Model received improvements in their accuracy.



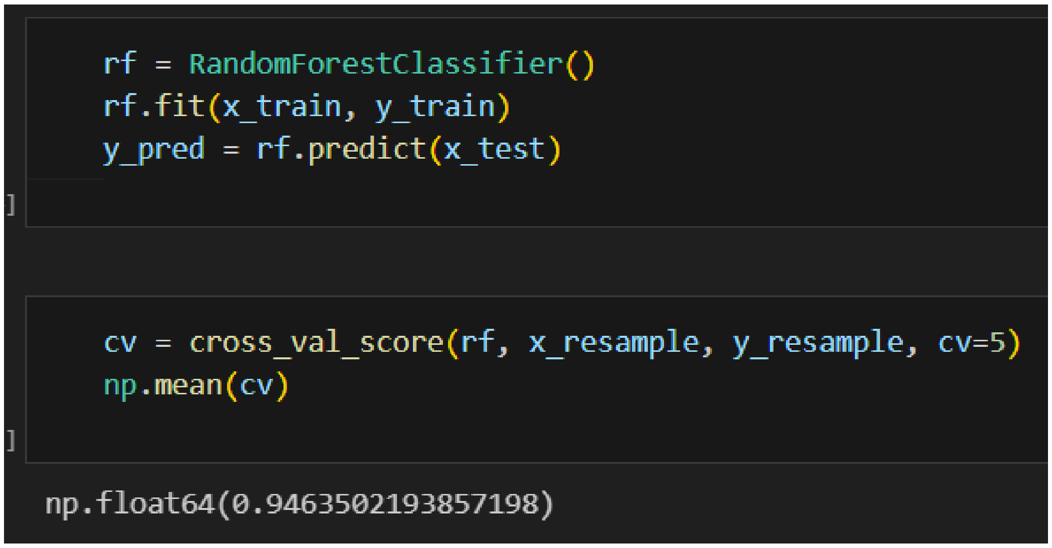
```
Tuned KNN Classifier
Best Parameters: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Confusion Matrix:
[[12960  2105]
 [  716 14303]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.86      0.90     15065
           1       0.87      0.95      0.91     15019

    accuracy                           0.91     30084
   macro avg       0.91      0.91      0.91     30084
weighted avg       0.91      0.91      0.91     30084
```



```
Tuned XGBoost Classifier (sklearn version)
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Confusion Matrix:
[[14157   908]
 [ 1704 13315]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.92     15065
           1       0.94      0.89      0.91     15019

    accuracy                           0.91     30084
   macro avg       0.91      0.91      0.91     30084
weighted avg       0.91      0.91      0.91     30084
```

Activity 3: Choosing best model and evaluating

The Random Forest Model is the best model.



```
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred = rf.predict(x_test)
```

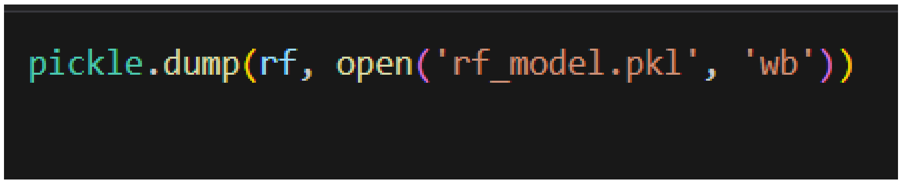
```
cv = cross_val_score(rf, x_resample, y_resample, cv=5)
np.mean(cv)
```

```
np.float64(0.9463502193857198)
```

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance.This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.



```
pickle.dump(rf, open('rf_model.pkl', 'wb'))
```

Activity 2: Integrate with Web Framework

We will be building a web application that is integrated to the model we built. A UI is provided to enter the values for predictions. The entered values are fed to the saved model and prediction is showcased on the webpage.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building HTML Pages

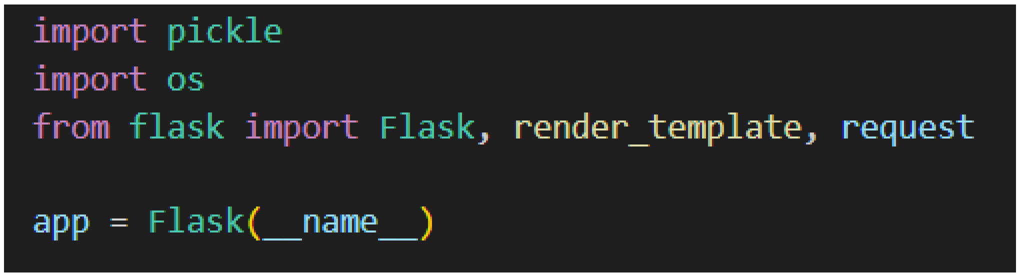
We create 4 HTML Pages namely

1. home.html
2. about.html
3. predict.html
4. submit.html

These are saved in the Flask->Templates folder. Refer the Github link for the file content.

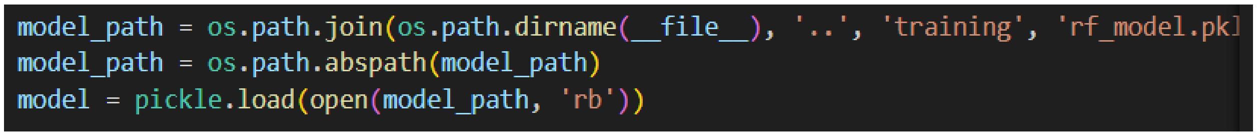
Activity 2.2: Building server-side script

First we import libraries and initialize the app



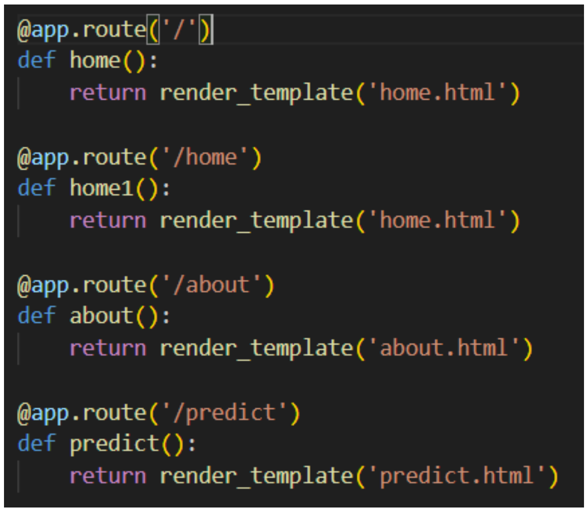
```
import pickle
import os
from flask import Flask, render_template, request

app = Flask(__name__)
```

Then we load the model



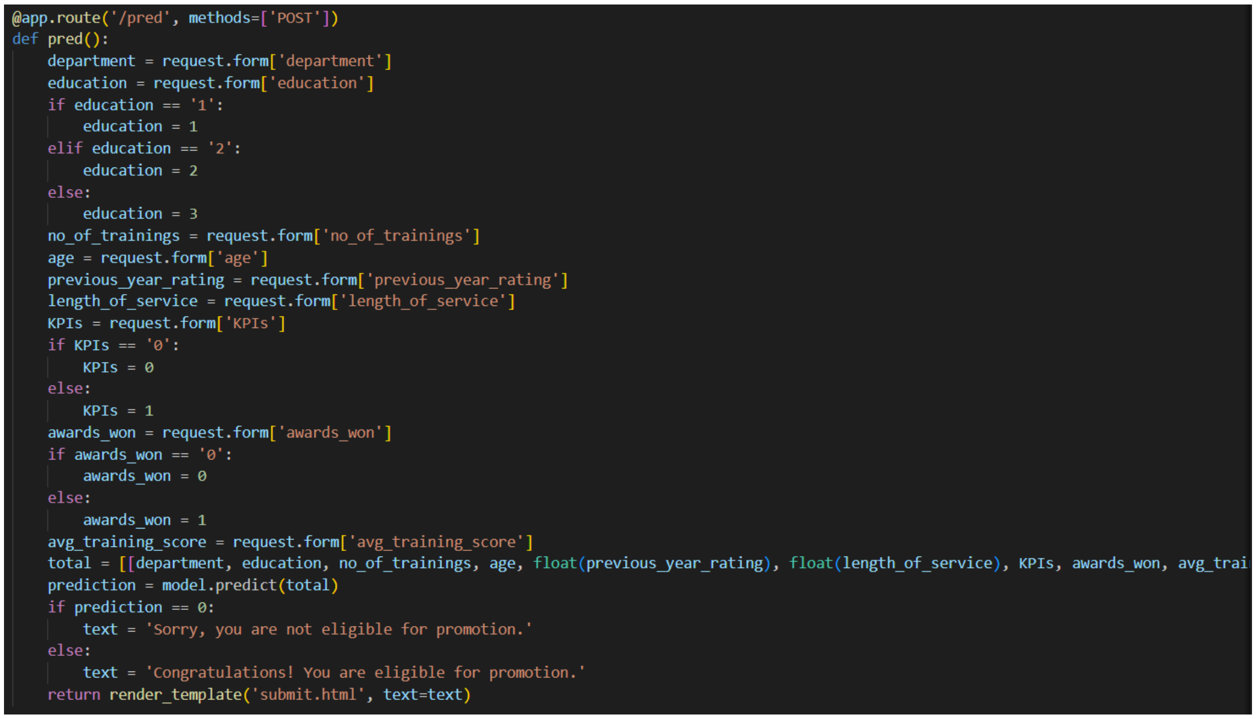
```
model_path = os.path.join(os.path.dirname(__file__), '..', 'training', 'rf_model.pkl
model_path = os.path.abspath(model_path)
model = pickle.load(open(model_path, 'rb'))
```

Now we define routes for the web application



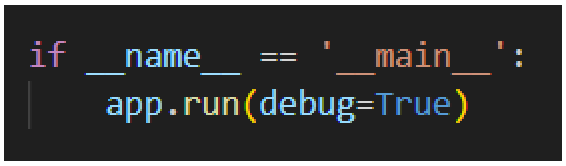
```
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/home')
def home1():
    return render_template('home.html')

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/predict')
def predict():
    return render_template('predict.html')
```

The prediction route is what gets the input from the user, processes and formats it. This then uses the loaded model to predict promotion eligibility.



```
@app.route('/pred', methods=['POST'])
def pred():
    department = request.form['department']
    education = request.form['education']
    if education == '1':
        education = 1
    elif education == '2':
        education = 2
    else:
        education = 3
    no_of_trainings = request.form['no_of_trainings']
    age = request.form['age']
    previous_year_rating = request.form['previous_year_rating']
    length_of_service = request.form['length_of_service']
    KPIs = request.form['KPIs']
    if KPIs == '0':
        KPIs = 0
    else:
        KPIs = 1
    awards_won = request.form['awards_won']
    if awards_won == '0':
        awards_won = 0
    else:
        awards_won = 1
    avg_training_score = request.form['avg_training_score']
    total = [[department, education, no_of_trainings, age, float(previous_year_rating), float(length_of_service), KPIs, awards_won, avg_trai
    prediction = model.predict(total)
    if prediction == 0:
        text = 'Sorry, you are not eligible for promotion.'
    else:
        text = 'Congratulations! You are eligible for promotion.'
    return render_template('submit.html', text=text)
```

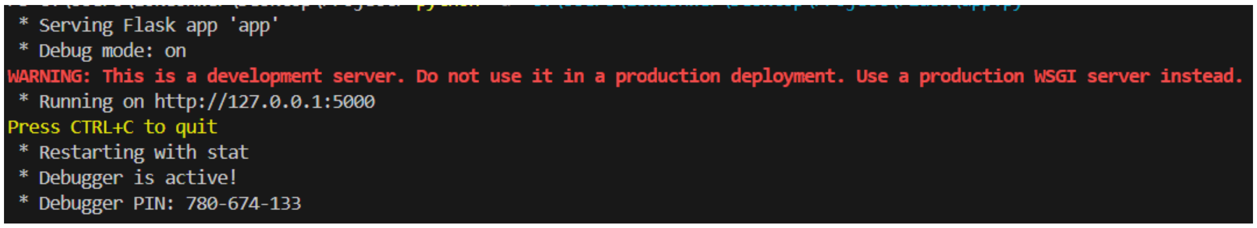
Main function (App runner)



```
if __name__ == '__main__':
    app.run(debug=True)
```

Activity 2.3: Run the web application

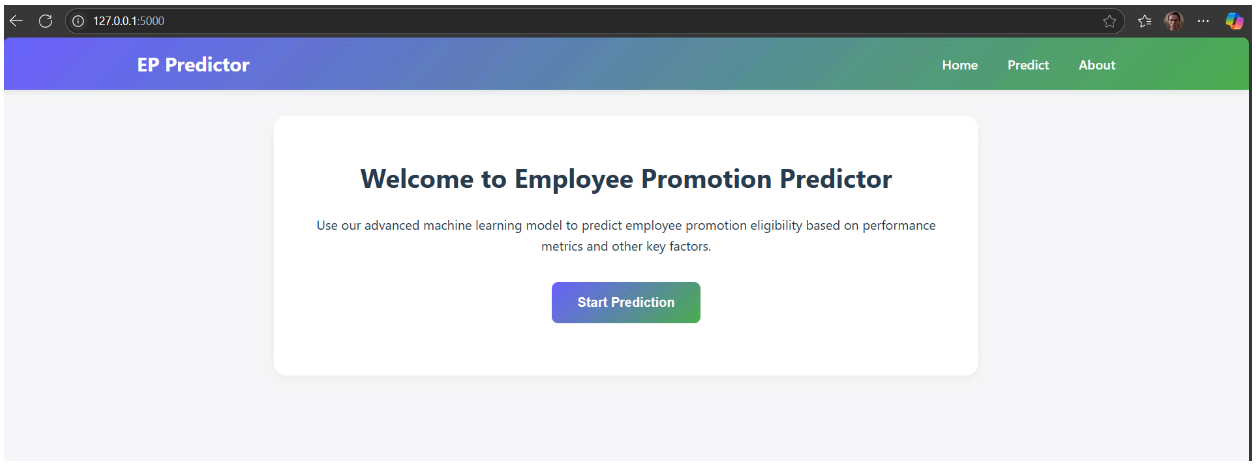
Open the ‘app.py’ in VSCode and run the program through python
The terminal displays your app status



```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 780-674-133
```

Go to your browser and write the localhost url (http://127.0.0.1:5000) to open the application.

Home Page


127.0.0.1:5000

EP Predictor

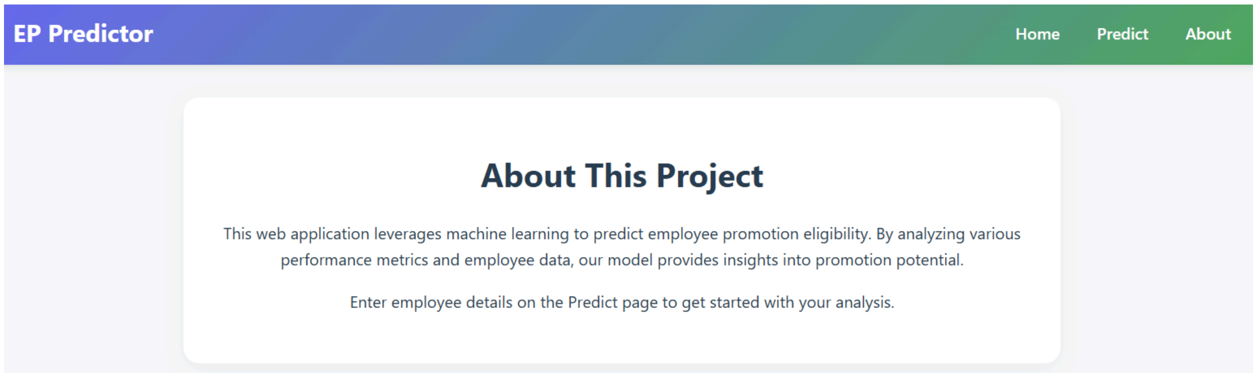
Home Predict About

Welcome to Employee Promotion Predictor

Use our advanced machine learning model to predict employee promotion eligibility based on performance metrics and other key factors.

Start Prediction

About Page


EP Predictor

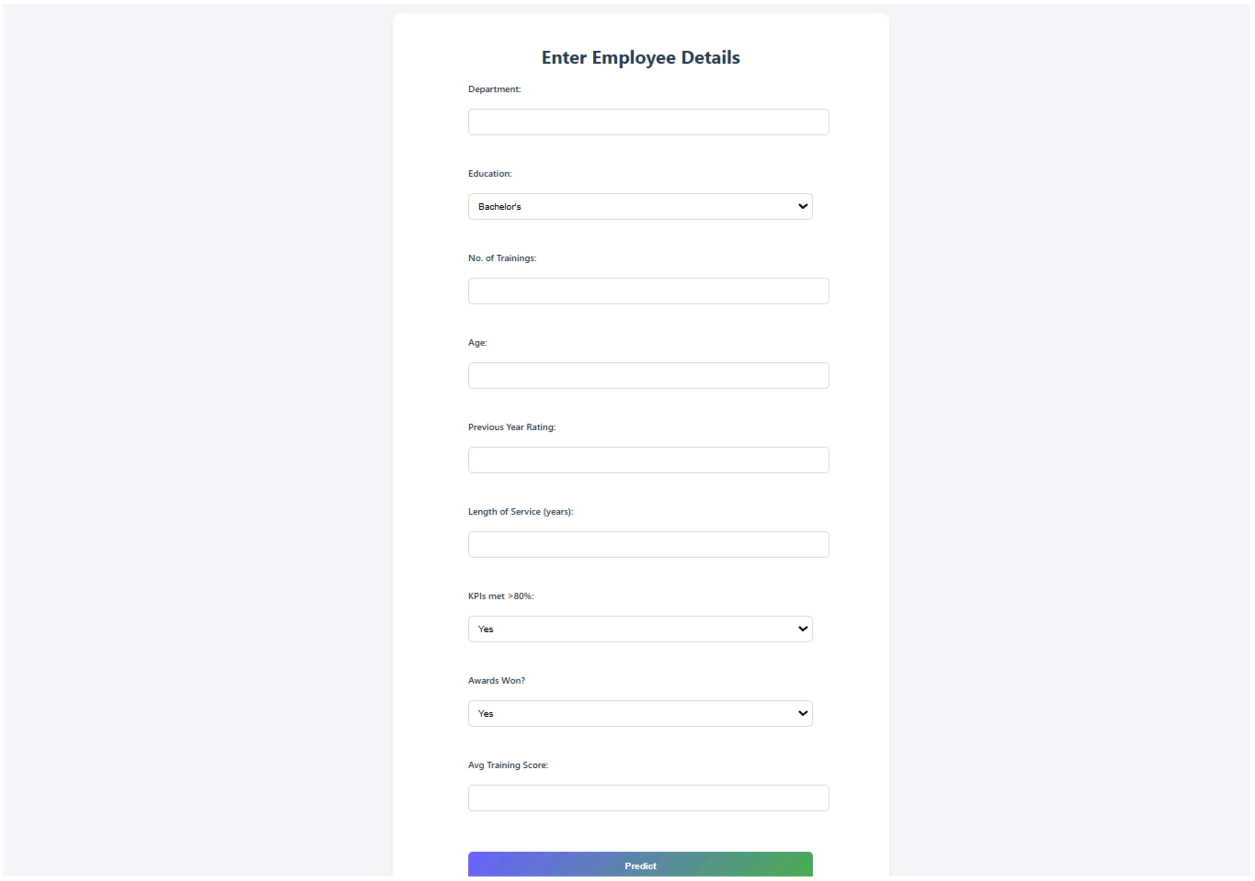
Home Predict About

About This Project

This web application leverages machine learning to predict employee promotion eligibility. By analyzing various performance metrics and employee data, our model provides insights into promotion potential.

Enter employee details on the Predict page to get started with your analysis.

Predict Page



Enter Employee Details

Department:

Education:

Bachelor's

No. of Trainings:

Age:

Previous Year Rating:

Length of Service (years):

KPIs met >80%:

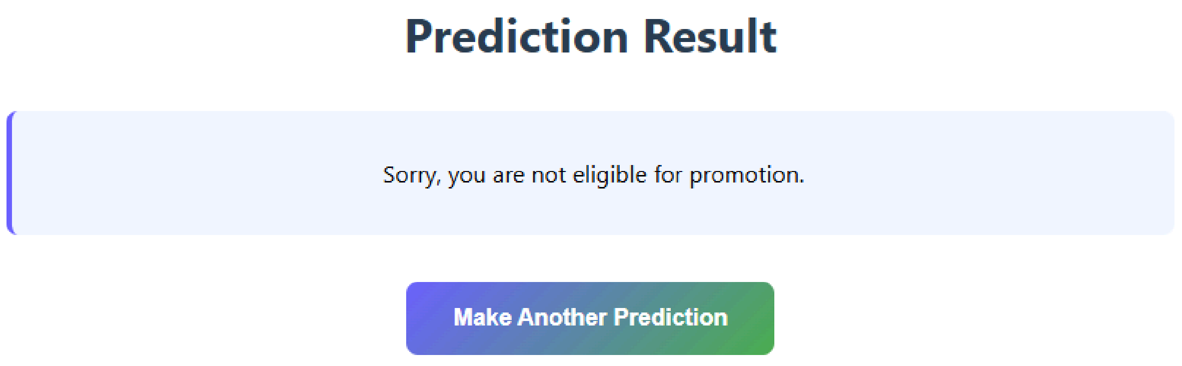
Yes

Awards Won?

Yes

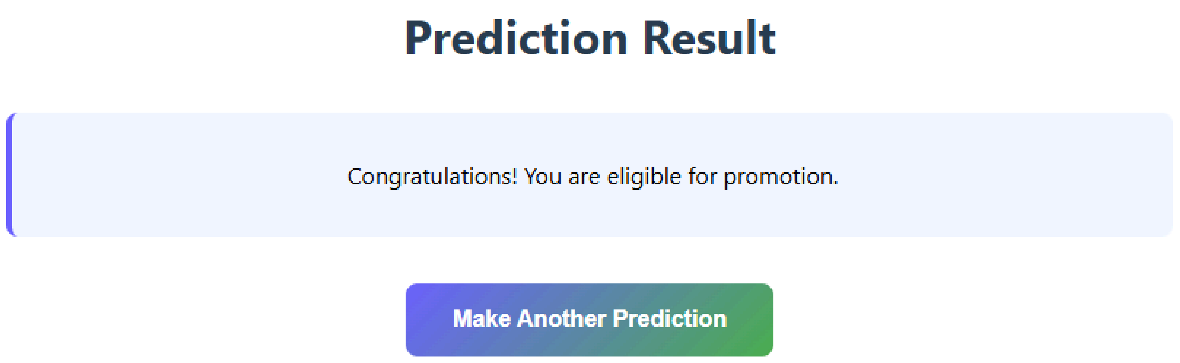
Avg Training Score:

Submit Page



Prediction Result

Sorry, you are not eligible for promotion.


Prediction Result

Congratulations! You are eligible for promotion.

Milestone 7: Project Demonstration and Documentation

Activity 1: Project Documentation

Refer Link.

Activity 2: Project Video Demonstration

Refer ‘6.Project Documentation and Demonstration’ in Link.